

CSE 410 Fall 2026
Privacy-Enhancing Technologies

Marina Blanton

Department of Computer Science and Engineering
University at Buffalo

Lecture 7: Protecting Data in Transit

Protecting Data in Transit

- We discussed many aspects of protecting data at rest
- Similar mechanisms are used for protecting data in transit
- The biggest missing piece is how to distribute keys for symmetric cryptography over the insecure internet
- This requires additional tools

Key Distribution Mechanisms

- Assume users communicate over an insecure network
- There are different possibilities for **key distribution**
 - they fundamentally differ based on whether the participants are within the same administrative domain
- When users are within the same administrative domain, we can rely on a trusted authority (TA)
 - the TA can verify user identities, issue certificates, transmit keys, etc.

Key Distribution Mechanisms

Major categories of key distribution are

- **key agreement** (a.k.a. key establishment or key exchange)
 - network users employ an interactive protocol to construct a session key
- **key predistribution**
 - a TA distributes keying information during the setup phase using a secure channel
 - a pair of users compute a key known only to them
- **session key distribution**
 - on request, an online TA chooses a session key and distributes it to two users
 - the TA communicates the new keys by encrypting them using previously distributed secret keys

Key Distribution Mechanisms

The **difference** between **key distribution** and **key agreement**:

- in **key distribution**, one party (a TA) chooses a key and transmits it to one or more parties
 - key transmission is performed in an encrypted form
- in **key agreement**, two or more parties jointly establish a secret key
 - communication is performed over a public channel
 - each participant contributes to the value of the resulting key
 - the key is not sent from one party to another

We'll talk about key agreement (key exchange) next

- this is achieved by means of **public-key cryptography**

Public-Key Cryptography

Public-key encryption

- a party creates a **public-private key pair** (pk, sk)
- the public key pk is used for encryption and is publicly available
- the private key sk is used for decryption only

$$\text{Dec}_{sk}(\text{Enc}_{pk}(m)) = m$$

- knowing the public key and the encryption algorithm only, it is computationally infeasible to find the secret key
- public-key crypto systems are also called **asymmetric**

Public-Key Cryptography

Digital signatures

- a party generated a public-private signing key pair (pk, sk)
- private key sk is used to sign a message
- public key pk is used to verify a signature on a message
- can be viewed as one-way message authentication

(Public-key) Key agreement or key distribution

- prior to the protocols the parties do not share a common secret
- after the protocol execution, they hold a key not known to any eavesdropper

Background

Before we proceed with the computation, we need to discuss the setup

Groups are a convenient way to represent sets and work with them

Background

A **group** G is a set of elements together with a binary operation \circ such that

- the set is **closed under the operation** \circ , i.e., for every $a, b \in G$, $a \circ b$ is a unique element of G
- the **associative law holds**, i.e., for all $a, b, c \in G$,
 $a \circ (b \circ c) = (a \circ b) \circ c$
- the set has a **unique identity element** e such that
 $a \circ e = e \circ a = a$ for every $a \in G$
- every element has a **unique inverse** a^{-1} in G such that
 $a \circ a^{-1} = a^{-1} \circ a = e$

Groups

Size of a group

- a group is **finite** if it has only a finite number of elements
- a group is **infinite** if it has an infinite number of elements
- the number of elements of a finite group is called the **order** of the group

Groups

Size of a group

- a group is **finite** if it has only a finite number of elements
- a group is **infinite** if it has an infinite number of elements
- the number of elements of a finite group is called the **order** of the group

Of particular interest to us are **multiplicative groups modulo a prime**

Groups

Example

- fix a positive integer m as the modulus
- the set of integers mod m relatively prime to m forms a group with multiplication modulo m as operation
- the identity element is 1
- it is called the **multiplicative group modulo m**

Groups

Example

- fix a positive integer m as the modulus
- the set of integers mod m relatively prime to m forms a group with multiplication modulo m as operation
- the identity element is 1
- it is called the **multiplicative group modulo m**
- let the modulus m be prime
- what is the group order?

More on Groups

- If a is an element of a finite group with identity 1, then there is a unique smallest positive integer i with $a^i = 1$
 - such i is called the **order of a** (different from the order of the group)
- A **cyclic group** is one that contains an element a whose powers a^i and a^{-i} make up the entire group
- An element a with such property is called a **generator** of the group

Cyclic Groups

Examples

- consider the multiplicative group modulo 7 (\mathbb{Z}_7^*, \cdot)
 - what is the order of 3?
 - what is the order of 2?
 - what is the order of 6?
- what is the multiplicative group generated by 3 modulo 11?

A group generator is often denoted by g

Discrete Logarithm Problem

Discrete logarithm: simple version

- given an appropriate group with generator g and $h = g^x$, it is difficult for someone to compute x
- x is called the **discrete logarithm** of h to the base g

Discrete Logarithm Problem

Discrete logarithm: simple version

- given an appropriate group with generator g and $h = g^x$, it is difficult for someone to compute x
- x is called the **discrete logarithm** of h to the base g

Discrete logarithm: more elaborate version

- we are given a cyclic group G of order q and its generator g
- for each $h \in G$ there is a unique x such that $g^x = h$
- such x is called the discrete logarithm of h with respect to g , written as $x = \log_g h$

Discrete Logarithm Problem

The discrete logarithm problem

- in a cyclic group G with given generator g , compute unique $\log_g h$ for a random element $h \in G$

Many properties of regular logarithms apply

- $\log_g 1 = 0$
- $\log_g(h_1 \cdot h_2) = (\log_g h_1 + \log_g h_2) \bmod q$

Discrete Logarithm Problem

We want the discrete logarithm problem to be **hard** (relative its setup)

When is it hard?

- most often a multiplicative group modulo prime p , \mathbb{Z}_p^* , or its subgroup is used
- the order of the group must contain a large prime
- the choice of parameters is driven by known algorithms for solving the discrete logarithm problem

Diffie-Hellman Key Exchange

Diffie-Hellman key exchange properties

- Alice and Bob compute the same key
- it is computationally difficult for someone else (without a or b) to compute their key
- the security property holds only against a **passive attacker**
 - a passive adversary simply monitors network communication
- a more powerful adversary can do severe damage

Threat Models

- Since the network is insecure, we need to protect against attackers
 - the adversary might be one of the users in the network
- An active adversary can:
 - prevent a message from being delivered
 - modify messages being transmitted on the network
 - try to masquerade as another user in the network
- Adversary's goal might be:
 - fool someone into accepting an invalid key as valid
 - learn some information about the key being established
 - use another user's identity to establish a shared key with someone

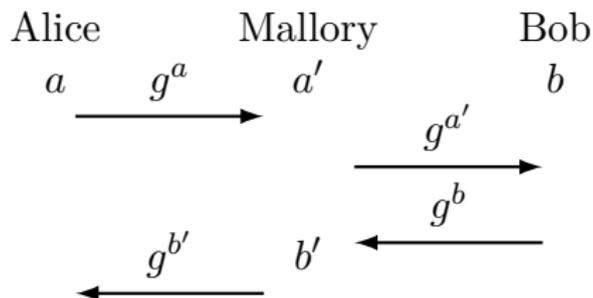
Diffie-Hellman Key Exchange

Security properties of Diffie-Hellman key exchange fall apart in the presence of an active adversary

- the attack is called a **man-in-the-middle attack**
- Mallory will intercept messages between Alice and Bob and substitute her own
- Alice establishes a shared key with Mallory and Bob also establishes a shared key with Mallory

Diffie-Hellman Key Exchange

Man-in-the-middle attack on Diffie-Hellman key exchange



- Alice shares the key $g^{ab'}$ with Mallory
- Bob shares the key $g^{a'b}$ with Mallory
- Alice and Bob do not share any key
- what is Mallory capable of doing?

Diffie-Hellman Key Exchange

Alice and Bob need to make sure they are exchanging messages with each other

- there is a need for **authentication**
- preceding this protocol with an authentication scheme is not guaranteed to solve the problem
 - after they authenticate, the same attack can be carried out

Diffie-Hellman Key Exchange

Alice and Bob need to make sure they are exchanging messages with each other

- there is a need for **authentication**
- preceding this protocol with an authentication scheme is not guaranteed to solve the problem
 - after they authenticate, the same attack can be carried out

We need a protocol that authenticates the participants at the same time the key is being established

- such a protocol is called an **authenticated key agreement scheme**
- it should simultaneously guarantee **secure mutual authentication** and **secure key computation**

Authenticated Diffie-Hellman Key Exchange

Authenticated Diffie-Hellman key exchange uses

- digital signatures
- certificates

We need to discuss those topics next

Digital Signatures

- A **digital signature scheme** is a method of signing messages stored in electronic form and verifying signatures
- Digital signatures can be used in very similar ways conventional signatures are used
 - paying by a credit card and signing the bill
 - signing a contract
 - signing a letter
- Unlike conventional signatures, we have that
 - digital signatures are not physically attached to messages
 - we cannot compare a digital signature to the original signature

Digital Signatures

Digital signatures allow us to achieve the following security objectives:

- authentication
- integrity
- non-repudiation
 - this is the main difference between signatures and MACs

What security property do we want from a digital signature scheme? How does it relate to that of MACs?

Digital Signatures

It is meaningful to consider the following **attack models**

- key-only attack
- known message attack
- chosen message attack

Adversarial goals might be

- total break
- selective forgery
- existential forgery

As with MACs, we want **existential unforgeability under an adaptive chosen message attack**

Digital Signatures

A digital signature scheme consists of three algorithms:

- **key generation**: given a security parameter κ , creates a public-private key pair (pk, sk)
- **signing algorithm** takes a messages m and uses private signing key sk and output a signature σ
- **signature verification algorithm** takes a message m , a signature on it σ , and the signer's public key pk and outputs a yes/no answer

Signature Algorithms

- **RSA signature scheme**
 - relies on the difficulty of factoring large composite moduli and hashing
- **ElGamal signature scheme**
 - was published in 1985 and works in groups where the discrete logarithm problem is hard
- **Schnorr signature scheme**
 - modifies ElGamal signature scheme to sign a digest of a message in a subgroup modulo p
- **Digital Signature Algorithm (DSA)**
 - a signature standard adopted by NIST

Digital Signature Standard (DSS)

Digital Signature Standard (DSS) or Digital Signature Algorithm (DSA) was adopted as a standard in 1994

- its design was influenced by prior ElGamal and Schnorr signature schemes
- it assumes the difficulty of the discrete logarithm problem
- no formal security proof exists

Digital Signature Standard (DSS)

- DSS was published in 1994 as **FIPS PUB 186**
 - specified to hash the message using SHA-1 before signing
 - produced a 320-bit signature on a 160-bit hash
- later variants such as **FIPS PUB 186-4** (2013) supported different sizes
 - DSA could be used with a 1024-, 2048-, or 3072-bit modulus
 - the signature size is 320, 448, or 512 bits (2 values of the same bitlength as q)
- the current version **FIPS PUB 186-5** uses only elliptic curve variants
 - this means shorter key length (256–512 bits) and thus faster performance
 - underlying arithmetic changes but not the algorithm

Digital Signature Security

Thorough evaluation of security of a signature scheme is crucial

- often a message can be encrypted and decrypted once and long-term security for the key is not required
- signatures can be used on legal documents and may need to be verified many years after signing
- choose the key length to be secure against future computing speeds

There is fundamental difference between **short-lived session keys** and **long-term public signing keys**

Digital Signature Algorithm

The setup requires a group where the discrete logarithm problem is hard

- choose a large prime p
- there is another large prime q that divides $p - 1$
- the bitlength of p and q must be from predefined size pairs
- g is a generator of subgroup of \mathbb{Z}_p^* of order q
- we obtain a setup for the group (p, q, g)

Digital Signature Algorithm

Key generation

- let (p, q, g) be a group setup for the discrete log problem to be hard
- let $H : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ be a cryptographic hash function
- choose secret $x \in \mathbb{Z}_q$
- compute $h = g^x \pmod{p}$
- the **public key** is $pk = (H, p, q, g, h)$
- the **private key** is $sk = x$

Digital Signature Algorithm

Signing

- given a message $m \in \{0, 1\}^*$, public key $pk = (H, p, q, g)$, and secret key $sk = x$
- choose $y \in \mathbb{Z}_q^*$ uniformly at random
- compute the signature $\sigma(m) = (\sigma_1, \sigma_2)$, where

$$\begin{aligned}\sigma_1 &= (g^y \bmod p) \bmod q \quad \text{and} \\ \sigma_2 &= (H(m) + x\sigma_1)y^{-1} \bmod q\end{aligned}$$

- if $\sigma_1 = 0$ or $\sigma_2 = 0$, a new value of y should be chosen

Digital Signature Algorithm

Signature verification

- given a message $m \in \{0, 1\}^*$, signature $\sigma(m) = (\sigma_1, \sigma_2)$ and $pk = (H, p, q, g, h)$
- compute

$$e_1 = H(m)\sigma_2^{-1} \bmod q$$

$$e_2 = \sigma_1\sigma_2^{-1} \bmod q$$

- then test $(g^{e_1}h^{e_2} \bmod p) \bmod q \stackrel{?}{=} \sigma_1$
- output 1 (valid) iff verification succeeds

Digital Signature Algorithm

Performance is favorable for public-key cryptography

- but still significantly slower than that of symmetric key cryptography

Security stood the test of time